Secure extension of FPGA general purpose processors for symmetric key cryptography with partial reconfiguration capabilities

Lubos Gaspar, Viktor Fischer, Lilian Bossuet, Robert Fouquet

Secure extension of FPGA general purpose processors for symmetric key cryptography with partial reconfiguration capabilities

LUBOS GASPAR, VIKTOR FISCHER, LILIAN BOSSUET and ROBERT FOUQUET

Hubert Curien Laboratory, Jean Monnet University, member of University of Lyon

Abstract. In data security systems, general purpose processors (GPPs) are often extended by a cryptographic accelerator. The paper presents three ways of extending GPPs for symmetric key cryptography applications. Proposed extensions guarantee secure key storage and management even if the system is facing protocol, software and cache memory attacks. The system is partitioned into processor, cipher, and key memory zones. The three security zones are separated at protocol, system, architecture and physical levels. The proposed principle was validated on Altera NIOS II, Xilinx MicroBlaze and Microsemi Cortex M1 soft core processor extensions. We show that stringent separation of the cipher zone is helpful for partial reconfiguration of the security module, if the enciphering algorithm needs to be dynamically changed. However, the key zone including reconfiguration controller must remain static in order to maintain the high level of security required. We demonstrate that the principle is feasible in partially reconfigurable field programmable gate arrays (FPGAs) such as Altera Stratix V or Xilinx Virtex 6 and also to some extent in FPGAs featuring hardwired general purpose processors such as Cortex M3 in Microsemi SmartFusion FPGA. Although the three GPPs feature different data interfaces, we show that the processors with their extensions reach the required high security level while maintaining partial reconfiguration capability.

Keywords: Security, Cryptosystems, Hardware security, Crypto-processor, FPGA, Soft-core processors, NIOS II, MicroBlaze, Cortex, Partial reconfiguration

The work presented in this paper was realized in the framework of SecReSoC project number ANR-09-SEGI-013, supported by the French National Research Agency (ANR). Authors' addresses: L. Gaspar, V. Fischer, L. Bossuet and R. Fouquet, Department of Computer Science, Telecommunications and Image Processing, Laboratoire Hubert Curien, Jean Monnet University, member of University of Lyon, 42000 Saint-Etienne, France.

1 Introduction

Data security systems often implement computationally extensive parallel cryptographic functions and complex sequential algorithms. Sequential algorithms are mostly implemented by general purpose processors (GPP), while parallel functions are implemented by the coprocessor placed inside the same cryptographic module or logic device. This approach is frequent in asymmetric key cryptography [20], [17], and also in symmetric key cryptography [7], [11]. Some embedded systems that use a processor – coprocessor approach implement both symmetric and asymmetric key algorithms in the same device [14].

General purpose processors offer flexibility, but at the same time, they weaken the security of all the above mentioned solutions. In order to resist side-channel attacks [22], the keys must be changed/manipulated regularly using a key management protocol. When a general-purpose processor manipulates confidential keys, the keys are saved in clear in processor registers or in the cache memory and may thus be exposed to software attacks.

Bangerter et al. showed in [5], that small malicious software can monitor the cache memory during enciphering and the key can be recovered remotely within a few minutes. In order to counter software attacks, the authors in [4] used two processors executing different tasks at different security levels. They created two virtual zones inside the physical memory: a protected memory zone for private key storage and an unprotected zone for public data. However, as both zones were located in the same physical memory, certain types of attacks, such as protocol attacks [2] or timing violation attacks, were still possible.

It is clear that software attacks targeting confidential keys can be countered only if enciphering is performed independently of the GPP, e.g. in a hardware cipher, while the keys must be stored in a dedicated memory. However, if the keys pass to the cipher via processor in clear, they are vulnerable to attacks. In the solution proposed here, the processor can manipulate the keys only indirectly: they are read/written from/to the key memory via a cipher and the processor can never read them in clear.

Flexible cryptographic hardware can be useful for several reasons: 1) hardware systems must follow the evolution of cryptographic standards; 2) security services must ensure compatibility between institutions and countries that use different cryptographic protocols and primitives; 3) as new attacks appear, corresponding countermeasures must be implemented [8]. One of the most promising emerging technologies is partial reconfiguration, which is supported by some Xilinx [23] and Altera [1] field-programmable gate arrays (FPGAs).

For security reasons, the partial bitstream must be enciphered and authenticated. This concept was proposed by Bossuet et al. [6], Parelkar et al. [19] and Drimer [10]. Improved bitstream authentication using a physically unclonable function (PUF) was proposed by Simpson et al. [21]. Another system enabling reconfiguration of processor peripherals after deciphering and authenticating bitstream was proposed by Kepa et al. [16]. In [9], Devic et al. proposed a protocol to ensure the protection of device reconfiguration against reply attacks.

None of the above-mentioned works considered secure key management and storage. In this paper, we present a novel concept of GPP extension for symmetric key cryptography with partial reconfiguration capability of the security module while keeping secret keys safe. It is an extended version of the conference paper [13].

The paper is organized as follows: In Section 2, we describe and discuss the creation of hardware security zones aimed at secure key management in conjunction with GPPs. In Section 3, we describe the novel principle of the security extension with indirect key management. In Section 4, we analyze the feasibility and security of partial device reconfiguration in the context of created security zones. In Section 5, we analyze and illustrate three basic ways of interfacing the security module with common GPPs in NIOS II, MicroBlaze and Cortex M1 processors. We present the results of implementations of the three architectures including partial reconfiguration capability in Section 6 and discuss them in Section 7. In Section 8 we present our conclusions.

2 Principle of separation of protected and unprotected zones

In the previous section, we explained that to resist software attacks on embedded systems using GPPs, the processor should not have access to confidential keys in clear. It is thus necessary to isolate it from the key memory at several levels.

2.1 Separation at protocol level

A cryptographic protocol aimed at data exchange in cryptographic applications must be robust against attacks. The keys to be exchanged must be encrypted and authenticated. If the received keys were deciphered in the processor, they could be exposed to software attacks. Therefore, they should be deciphered outside the processor in a dedicated unit and never leave the unit in clear. Once the keys are deciphered and authenticated, they can be used for data enciphering/deciphering and authentication, but still outside the processor. However, enciphered/deciphered data blocks can be processed directly by the processor, e.g. when performing cipher mode operations.

It is the protocol that has to clearly separate key management and data processing tasks and to define how and by which blocks the tasks are performed. The protocol must also define the structure of the keys. If a hierarchical key structure is used, higher-level keys are used to encipher and authenticate lower-level keys. The low-level session keys are used for data enciphering, deciphering and authentication.

2.2 Separation at system level

The principle of the separation at system level is illustrated in Fig. 1. This principle is based on the creation of three zones: a processor zone, a cipher zone

and a key zone. The processor exchanges data with the cipher across the data bus (large bus in black in Fig. 1). Encrypted session keys are also transported through this data bus when being exchanged with other communication counterparts.

Keys are stored in clear in a dedicated memory situated in the key zone. The key memory has a hierarchical structure and is separated from the GPP by the cipher. All the keys are transferred between the cipher and the key memory via the key memory bus (in gray in Fig. 1), except for master keys, which are introduced into the memory via a separate input during device initialization. The key initialization bus must be separated from the data bus (in black) that connects the GPP with the cipher. It is crucial that no paths that allow secret keys to pass in clear from the key memory bus to the data bus exist. This condition is very important for security because it guarantees separation of the key and processor zones.

Before enciphering/deciphering data blocks and keys, the cipher is initialized with the selected key (session key or higher level key) via the cipher key bus (in white in Fig. 1). Key selection is controlled by the processor through a control bus. The key address space must be completely covered – no unused key address can remain.

The principle of creating security zones is independent from the type of enciphering algorithm – any symmetric key block cipher with or without side channel attack countermeasures can be used. Furthermore, the fence separating the key zone from the cipher and processor zones can be used during partial reconfiguration of the system when updating the processor or the cryptographic algorithm, while maintaining the key memory contents unchanged.

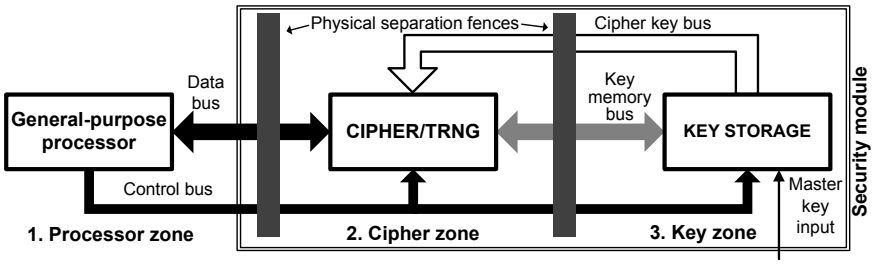

Fig. 1. Separation at system level

2.3 Separation at architectural level

To achieve efficient separation of the security zones, the data bus, key memory bus and cipher key bus cannot cross more than one security fence. Bus multiplexers directing the flow of data must be placed in such a way that, even if their control is violated, no physical path can be created for keys to escape from the key zone.

Interfaces between partitions must be designed to respect special constraints: data buses must be unidirectional and the communications must be controlled only by one module (GPP in our case). A straightforward interface design simplifies implementation of bus macros in partial reconfiguration design flow.

2.4 Separation at physical level

According to NSTISS guidelines [18], physical separation of security zones minimizes the possibility of the loss or corruption of secret keys by residual electromagnetic radiation from the protected zone (the key zone) to the unprotected zone (the processor zone). This guideline is followed in Xilinx Single Chip Crypto (SCC) design tools [24]. SCC tools require creation of an empty area (insulation fence) at the border of security zones (see Fig. 1). Only selected signals can cross the fence.

The SCC design flow is similar to that of partial reconfiguration: the design is divided into partitions, all partitions are compiled as top level entities, interfaces between partitions use special bus macros, etc. [24].

3 Implementation of the security module

The implementation of the security module is illustrated in Fig. 2. Three zones (the processor interface, the cipher, and the key zone) can be clearly distinguished. Three buses are used: a data bus (in black), a key memory bus (in gray) and a cipher key bus (in white). Separation rules are strictly applied: key buses never pass through the processor interface zone and data buses never pass through the key zone. Secret keys can never leave the key zone without passing through the cipher.

As presented in Sec. 2, any enciphering algorithm can be used in the security module. In order to validate the system architecture, we use a 128-bit Advanced Encryption Standard (AES) because it is the most common currently used algorithm.

Keys are organized in two hierarchical levels. High-level master keys for session key enciphering and authentication are stored in the master key register. These keys are initialized via a dedicated key input during system initialization by the trusted entity. Session keys are generated inside the module by a true random number generator (TRNG) and post-processed by the decipher core or received from the processor and deciphered and authenticated using master keys. Again, any TRNG principle can be used. Session keys are used only for data enciphering/deciphering (using cipher modes) and authentication (e.g. using CBC-MAC mode).

Since the security module complies with stringent separation rules, it is secure by design, and no software or protocol attacks can result in disclosure of secret keys. For this reason, any protocol can be implemented, while the key protection remains the same. Of course, the proposed solution will not resist protocol attacks that do not target secret keys, such as service denial attacks.

These should be dealt with at software level, which is beyond the scope of this paper.

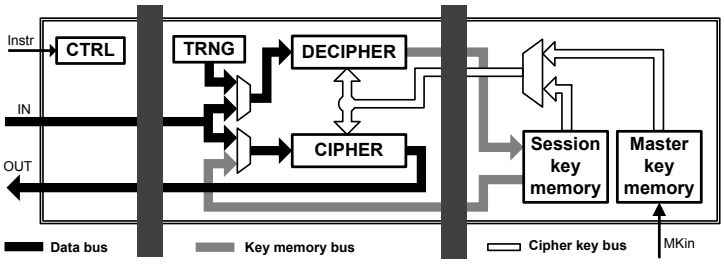

Fig. 2. Security module implementation

3.1 Example of a communication protocol

The cryptographic protocol for communication between A and B (see Fig. 3) illustrates the efficiency of the proposed structure. In practice, any other common cryptographic protocol can be implemented. Tasks 1, 2, 3, 8 and 9 are performed only by the security module, Tasks 5, 6, 7 are executed only by the GPP. Tasks 4 and 10 represent iterative implementation of encryption modes (EM) performed by the GPP (registering and xor-ing subsequent data blocks) and by the security module (enciphering E and deciphering E^{-1}). In this protocol, we assume that both devices were initialized by a trusted entity TE using the same enciphering (MK) and authentication (AMK) master keys. First, the device starting the communication (A), generates a new session key SK: the unprocessed session key is generated by the TRNG, post-processed cryptographically in the decipher using the MK master key and saved in clear in the session key memory (Task 1 in Fig. 3). Next, the session key SK is enciphered using master key MK and read by the processor (Task 2). Finally, a digital fingerprint FP_A is generated by enciphering SK using the authentication master key AMK (Task 3).

When both the session key SK and its fingerprint FP_A are generated, Task 4 can be executed in a loop: data blocks ($DATA_i$) are sent by the GPP to the cipher module, which enciphers them using SK and sends them back to the GPP as $CDATA_i$. The GPP combines input and output blocks according to the encryption mode algorithm (EM) and computes $MCDATA_i$. Finally, it creates the packet P containing the enciphered session key CSK, its fingerprint FP_A, and enciphered data blocks $MCDATA_i$ (Task 5). The packet is sent to device B (Task 6).

The processor in B receives the packet P (Task 7) and extracts the enciphered session key CSK and its digital fingerprint FP_A. The key is then sent to the security module, where it is deciphered using the master key MK and stored in the session key memory (Task 8). The security module generates a fingerprint

FP_B of the session key SK using the master authentication key AMK (Task 9) and sends it back to the processor, which compares it with the received fingerprint FP_A (Task 10). If FP_A and FP_B are the same, the session key is authenticated and can be used for data enciphering/deciphering (the loop in Task 10).

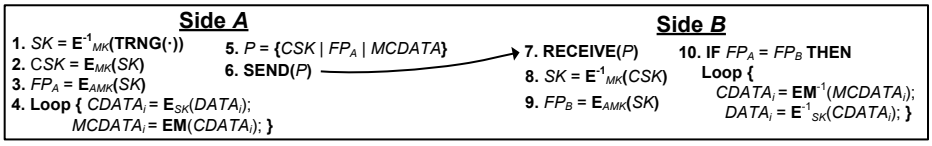

Fig. 3. Communication protocol between two devices (symbol | represents concatenation of blocks of data, E means encryption, EM means encryption mode, SK represents session key, MK master key, CSK enciphered session key, AMK authentication master key and FP is the fingerprint)

4 Reconfiguration of the security module

Upgrading the hardware is of particular interest in many cryptographic applications. Next, we analyze the benefits and implications of full and partial reconfiguration.

4.1 Total reconfiguration versus partial reconfiguration of the device

Cryptographic modules based on symmetric key cryptography must share the same cryptographic key. If the device has been totally reconfigured, the key must be reinitialized. This must be done in a secure environment. Remote key initialization is very dangerous, because the key cannot be enciphered without the other key and must consequently be transferred in clear. The initialization key cannot be included in the reconfiguration bitstream, because compromising one device would compromise the whole set of devices. Partial device reconfiguration is a better solution for single chip cryptographic applications: the master key can be initialized once in a protected environment and then stored in a static logic partition. During device upgrades, the key is kept the same and only partially reconfigurable blocks and the GPP software are allowed to be changed.

When dividing the system into static and reconfigurable partitions, the approach proposed in Sec. 2 for separating buses, key memory, cipher and processor zones is very useful, because communication interfaces between future static and reconfigurable partitions can be easily managed. There are two solutions to partitioning cryptographic system in Fig. 1: a) the key zone and processor zones are static and only the cipher zone is reconfigurable; b) the key zone (or

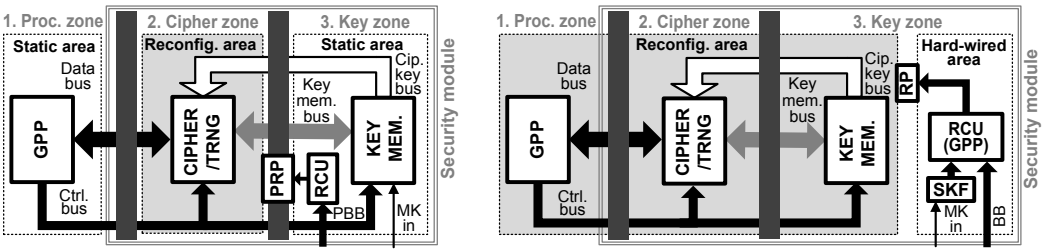

Fig. 4. Separation principle including partial reconfiguration capability with reconfiguration of the cipher zone only (left) and reconfiguration of the processor, cipher and key zone (right)

at least master key memory) is kept static and the remainder of the device is dynamically reconfigurable. In both solutions, the cipher zone belongs to the reconfigurable partition and can thus reflect the required algorithm changes. The static area containing confidential keys must be handled with special care to avoid unauthorized access to confidential keys after device reconfiguration by fake reconfiguration data.

4.2 Validation of the principle of security module partial reconfiguration in SRAM FPGAs

The first solution from the previous section can be implemented in partially reconfigurable Xilinx FPGAs such as Virtex 5 and 6 and recently also in Altera FPGAs (starting from the Stratix V family) as shown in the left part of Fig. 4. We selected the Xilinx Virtex 6 FPGA family for our tests, because software tools for partial reconfiguration for Altera technologies were not available.

In order to validate the separation concept, we propose three types of reconfigurable modules: A) one containing the AES cipher and decipher; B) the second one containing the DES cipher and decipher (note that the cipher and decipher must be separate, while in the DES algorithm, ciphering and deciphering can be performed by the same piece of hardware); C) the third one representing an empty black box module.

As required, reconfigurable modules A, B and C have the same interfaces with the surrounding static partition (containing GPP and key memory). The dimensions of the reconfigurable partition must be set to meet the requirements of the biggest module (the one containing the AES cipher/decipher in our case).

The reconfiguration of the reconfigurable logic area is controlled by the reconfiguration control unit (RCU) placed in the static partition. It is in charge of transferring new logic to the reconfigurable area via a partial reconfiguration port (PRP). The partial bitstream is transferred to the RCU via the partial bitstream bus (PBB). Since the RCU must also ensure (by verifying bitstream integrity and authenticity) that the reconfigurable partition was not modified by an unauthorized person, its implementation is crucial for security. Additional

techniques can be used to increase security (e.g. zeroization of all flip-flops and configuration bits in the reconfigurable area by the PRP before being configured using the new partial bitstream [15]). RCU can be implemented using another GPP, a dedicated processor or a state machine, but the reconfiguration controller must be strictly separate from the GPP controlling the communication channel. The executable code of the RCU must be write protected to resist software attacks. Since our objective was to demonstrate the separation concept, in this first case, we only implemented a simple reconfiguration controller (state machine) placed in the static logic area.

4.3 Reconfiguration of security modules in FPGAs containing hardwired GPPs

Many FPGAs are not partially reconfigurable. For devices containing hardwired GPPs featuring a non-volatile memory (e.g. MicroSemi SmartFusion FPGA), we propose another solution that combines the possibility of hardware upgrades with secure key management (see right part of Fig. 4). The hardwired GPP can serve as a reconfiguration controller and confidential keys (or at least master keys) can be saved in its non-volatile memory – secure key flash memory (SKF). In this case, the entire programmable logic fabric can serve as a reconfigurable area containing all the system blocks (including the second GPP) except for the reconfiguration controller and the confidential key memory. The bitstream is transferred into RCU via the bitstream bus (BB) and the logic area is configured via the reconfiguration port (RP).

Note that all security zones and especially processor and cipher zones must remain separate as required in Sec. 2. It is also of paramount importance that the use of the hardwired GPP should be strictly limited to reconfiguration tasks. In particular, it must not be connected to the data bus of the second GPP that is in charge of data processing, i.e. the separation principle must be maintained.

Although this solution is less flexible and slower than the one discussed above, it still offers secure key management and high-level protection of confidential keys. However, the use of the powerful hardwired processor for device reconfiguration will exclude it from other tasks (and especially from communication with the cipher and from key management). Another GPP will have to be used and implemented in the reconfigurable zone, what makes this solution less attractive. For this reason, we did not implement it in hardware.

5 Interfacing embedded processors with the security module

There are three possible ways to connect the security module with the embedded processor: 1) the security module can be included in the processor's data path; 2) the security module can be accessible via the internal register file of the processor; 3) the security module can be accessible as a peripheral via the peripheral bus. In each case, the security module remains the same and it is completed by a wrapper

that is compatible with the processor's interface. The wrapper translates control commands and converts the bus width.

5.1 Including security module in the processor's data path

In this case, security module operations are implemented as custom instructions of the processor. Since the module is included in the processor's data path, it affects its maximal clock frequency. This kind of GPP and security module interconnection was implemented in Altera NIOS II processor, as illustrated in the left part of Fig. 5.

Because of the use of custom instructions, the NIOS II control unit directly drives the operation of the security module, thus avoiding unwanted latency increase. The point-to-point connection between the processor and its security module ensures that the communication is secure (it cannot be eavesdropped by some other peripheral).

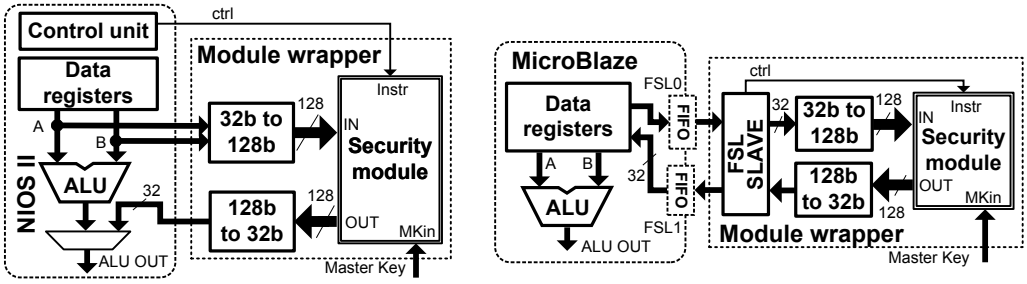

Fig. 5. Interfacing NIOS II (left) and MicroBlaze (right) with the security module

5.2 Interconnecting GPP with the security module using the processor's register file

In contrast with the previous solution, the instruction set does not need to be customized if the security module is connected to the processor's register file via a dedicated internal bus. However, this operation requires execution of an additional instruction in the program code. This slows down code execution and data exchange between the processor and the security module. On the other hand, in this case, the security module is not included in the processor's critical path, so the clock frequency of the processor is not affected. The point-to-point connection via the dedicated bus ensures a high level of security for bus communications. This kind of interfacing between the GPP and the security module could be applied to the Xilinx MicroBlaze processor, as shown in the right part of Fig. 5.

The MicroBlaze architecture features the high-performance 32-bit Fast Simplex Link (FSL), aimed at interfacing external modules with processor registers. Unfortunately, the FSL standard does not define the control interface, so before each operation, a 32-bit control instruction has to be sent to the module via the FSL FIFOs. Despite the fact that FIFOs insert additional latencies, they separate the processor and security module clock domains, so that the security module can run at a higher clock frequency than the processor.

5.3 Accessing the security module as a peripheral

The most widely applicable solution for interfacing GPP with the security module is to access the module via a point-to-multipoint peripheral bus. This communication is less secure than the discussed point-to-point communications, but it is available for all GPPs. For example, this is the only solution that can be applied in Microsemi FPGAs featuring Cortex processors and AHB bus [3] as illustrated in Fig. 6.

Although the AHB bus does not include a control interface, the address bus can be used to pass commands to the security module in parallel with data. On the other hand, the AHB bus is shared among several bus slaves (program flash memory, RAM, etc.), therefore the data exchange rate with the security module is slowed down.

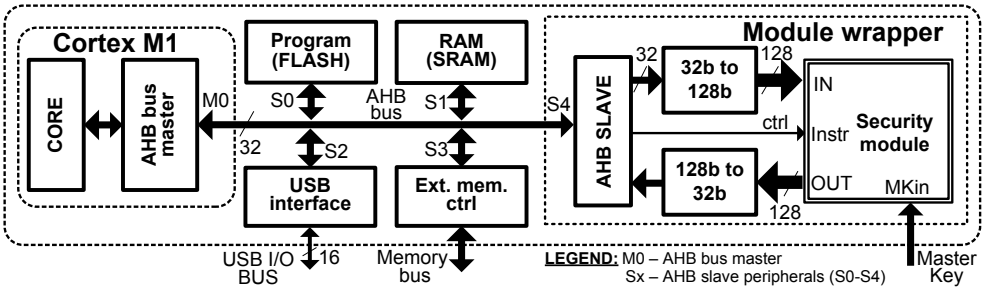

Fig. 6. Interfacing Cortex M1 with the security module

6 Results

The three processors with their security extension modules were described in VHDL and mapped to three FPGA families. The NIOS II system was mapped in Stratix II EP2S60F672C5ES device (NIOS II evaluation board) using Quartus II (ver. 9.2). The MicroBlaze system was mapped in Virtex 6 XC6VLX240TFF1156 device (Xilinx ML605 evaluation kit) using ISE (ver. 12.4). The Cortex M1 system was mapped in Fusion M1AFS1500-FGG484 device (Microsemi Fusion

Table 1. Utilization of FPGA resources by processors with the security module containing the AES cipher

	NIOS II		Cortex M1		MicroBlaze		Reconf. MBlaze	
	ALMs	**RAM kb**	**Tiles**	**RAM kb**	**Slices**	**RAM kb**	**Slices**	**RAM kb**
System total	2531	243.9	15053	216.0	1954	1206.0	2619	1206.0
→ Processor	1204	187.9	9433	104.0	1350	774.0	1350	774.0
→ Sec. module	1327	56.0	5620	112.0	604	432.0	1269	432.0
Ext. overhead	110.2%	29.8%	59.6%	107.7%	44.7%	55.8%	94.0%	55.8%

Table 2. Comparison of three versions of the reconfigurable module (RM) in an extended MicroBlaze system

	Static	Reconfigurable partition			Total
	part.	**AES**	**DES**	**Empty black box**	**with AES RM**
Slices	1976	643	442	264	2619
RAM kb	846	360	0	0	1206

embedded development kit) using Libero (ver. 8.5, SP2). A hardware module including the Cypress USB device CY7C68013A was connected to evaluation boards for data transfers from/to the PC.

The results of the implementation are given in Table 1. The area is represented by number of Adaptive Logic Modules – ALMs (for Altera), Slices (for Xilinx) and Tiles (for Microsemi family). Unfortunately, the results cannot be directly compared, because ALMs, Slices and Tiles have different internal structure, inputs and flip-flop counts.

Columns 8 and 9 in Table 1 list the resources of a partially reconfigurable system based on MicroBlaze extended by the reconfigurable module with AES cipher and TRNG. The reconfiguration overhead can be observed by comparing the MicroBlaze system with the static and dynamically configurable security module. Individual results for three different reconfigurable modules in the extended MicroBlaze system are listed in Table 2. Note that the size of the system including both the static and reconfigurable partition containing AES in Table 2 is the same as the sum of resources occupied by the processor and the security module in columns 8 and 9 in Table 1. However, the security module in column 8 of Table 1 occupies more slices (1269) than the reconfigurable AES module in Table 2 (643), because the key memory and the wrapper of the second module are included in the static partition.

The floor plan of the MicroBlaze GPP linked with the security module containing the cipher zone and the key memory zone is depicted in Fig. 7. The three regions are separated by large spaces (fences) according to the SCC requirements.

In order to compare throughput, the clock frequency of all three systems was fixed to 50 MHz. The throughput was evaluated by transferring packets from the PC to the FPGA (and vice versa) via a USB interface. Each packet contained an encrypted session key, its digital fingerprint and five 128-bit payload blocks. The communication protocol is explained in Sec. 3.1. When implementing

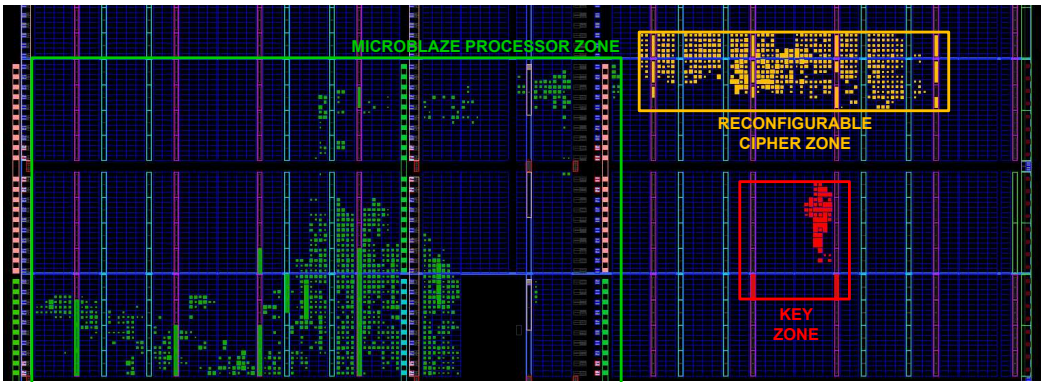

Fig. 7. Floor plan of the system containing three separate security zones: reconfigurable cipher, key memory and MicroBlaze GPP

this protocol, the NIOS II-based system achieved an overall throughput of 25.1 Mb/s, the MicroBlaze-based system achieved 18.4 Mb/s and the Cortex M1 system achieved 12.2 Mb/s. Differences in throughput are mainly due to different interfaces between the GPP and the cryptocoprocessor.

7 Discussion

In the Altera FPGA, the security module area is similar to that of the GPP (1327 vs. 1204 ALMs giving 110%). In the Xilinx family, the security module appears to be smaller (the area overhead is only 45%). This is due to the size of the processor and not directly to that of the module. Implementation in the Xilinx family appears to use much more memory. In the case of the security module, it is because the RAM blocks are bigger than in the other two FPGA families and they are not optimally utilized (i.e. two 8-bit S-boxes are mapped into one 18-kbit block, but only use 2 kbits). The MicroBlaze processor's memory requirements are higher, because of the size of the MicroBlaze register file and because of the use of FIFOs that must be used to connect the security module with the FSL bus. In the Cortex M1 system, the security module area overhead is 60%. In all three solutions, the cost of the GPP security extension is related to the size of the cipher and memory modules. The cost of the isolation of security zones is negligible.

For partial reconfiguration of Xilinx FPGAs, partition pins increase the occupied area. This effect can be observed if an empty reconfigurable block is implemented. Note that five 128-bit buses and associated control signals must cross the reconfigurable partition border and each slice can implement at most four partition pins. Since not all slices are fully utilized, the number of slices in the empty black box is higher than required. The system featuring partially reconfigurable AES cipher is bigger than that using a fixed AES module, because the separate design regions are compiled independently. This second effect can be

observed by comparing two versions of the extended MicroBlaze system (with or without partial reconfiguration of the cipher zone): the area extension overhead with the AES cipher is 94.0% versus 44.7%. Note that partial reconfiguration also increases the latency of the system. However, we assume that reconfiguration is only performed occasionally (e.g. if a new attack countermeasure has to be applied). The timing overhead of the reconfiguration is thus negligible.

The MicroBlaze processor with its extension achieves 73% of the throughput of the NIOS II. This is due to the FSL bus protocol, compared with the simple custom instruction implementation in NIOS. This difference could be reduced in the MicroBlaze system if data were transferred to the security module using a DMA. The FSL bus would only be used for transporting instructions to the security module. On the other hand, the Cortex M1 processor with its security module extension achieves only 49% of the throughput of the NIOS II implementation. This is because of the nature of the AHB bus which is shared among all communicating units.

It is clear that the architectures presented here could be further analyzed and optimized from the point of view of performance, area and power consumption. However, it is also clear that the security overhead due to the creation of isolation fences and due to the application of separation rules is negligible.

The proposed solution does not provide protection against physical attacks such as side channel attacks. We assume that such countermeasures will be included in the cipher module. Indeed, one of the main advantages of the partial reconfigurability of the device is that the cipher module can be updated by a new cipher version implementing countermeasures against the most recent attacks. This approach is not possible in hardwired architectures.

The principle presented in this paper can be extended to any GPP. One of solutions would be to use an open source GPP and to include the security module in the processor's data path as was the case with the NIOS II processor. This principle can be also applied to a specific-purpose processor such as that published in [12].

8 Conclusion

In this paper, we propose a novel cryptographic extension of GPPs manipulating secret keys in a highly secure way. The principle is based on the creation of separate processor, cipher and key zones. Separation is implemented at protocol, system, architectural and physical levels, and guarantees that unencrypted keys can never be transferred from the protected key zone to the unprotected processor zone. The only way to transfer the keys to the processor zone is to pass across the cipher zone: the keys are enciphered before entering the processor zone and must be deciphered when going in the opposite direction (i.e. when entering the memory zone). The proposed solution substantially enhances security compared with existing soft-core cryptographic extensions.

The separation principle was implemented in FPGAs and tested using NIOS II, MicroBlaze and Cortex M1 processors. The obtained throughput including

the processing of packets, key management and data enciphering/deciphering and authentication was about 25, 18 and 12 Mb/s, respectively. This speed was mainly limited by the processors and their data interfaces. The area of the system was increased by 110% compared with the smaller NIOS II processor, by 44% when the MicroBlaze processor was taken as a basis and by 60% when Cortex M1 was extended using the security module.

A partially reconfigurable security module was proposed for the extended MicroBlaze system. The system with a reconfigurable AES cipher is bigger than that using a fixed AES module: the area extension overhead of the first type of the module is 94.0% versus 44.7% of the second one. We also showed that stringent security requirements can be met in partially reconfigurable system only if the key zone is kept static.

References

1. Altera. *FPGA Run-Time Reconfiguration: Two Approaches*, 2010. http://www.altera.com/.
2. R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov. Cryptographic processors - a survey. *Proceedings of the IEEE*, 94(2):357–369, 2006.
3. ARM. *AMBA Specification, rev. 2.0*, 1999. http://www.arm.com.
4. A. Ashkenazi and D. Akselrod. Platform independent overall security architecture in multi-processor system-on-chip integrated circuits for use in mobile phones and handheld devices. *Computers & Electrical Engineering*, 33(5-6):407–424, 2007.
5. E. Bangerter, D. Gullasch, and S. Krenn. Cache games–Bringing access-based cache attacks on AES to practice. *Workshop COSADE*, pages 215–221, 2011.
6. L. Bossuet, G. Gogniat, and W. Burleson. Dynamically configurable security for sram fpga bitstreams. *Reconfigurable Architectures Workshop (RAW)*, pages 146–154, 2004.
7. F. Crowe, A. Daly, T. Kerins, and W. Marnane. Single-chip FPGA implementation of a cryptographic co-processor. In *FPT'04*, pages 279–285, 2004.
8. P. Davies. Flexibile Security. Thales e-Security, White Paper - Cryptography & Interoperability, August 2003.
9. F. Devic, L. Torres, and B. Badrignans. Secure protocol implementation for remote bitstream update preventing replay attacks on fpga. In *FPL'10*, pages 179–182, 2010.
10. S. Drimer. Authentication of fpga bitstreams: Why and how. *In Applied Reconfigurable Computing, volume 4419 of LNCS*, pages 73–84, 2007.
11. Y. Eslami, A. Sheikholeslami, P.G. Gulak, S. Masui, and K. Mukaida. An area-efficient universal cryptography processor for smart cards. In *2006 IEEE Transitions on VLSI systems*, pages 43–56, 2006.
12. L. Gaspar, V. Fischer, F. Bernard, L. Bossuet, and P. Cotret. HCrypt: A Novel Concept of Crypto-processor with Secured Key Management. *ReConFig'10*, pages 280–285, 2010.
13. L. Gaspar, V. Fischer, L. Bossuet, and R. Fouquet. Secure extensions of FPGA soft core processors for symmetric key cryptography. *ReCoSoC'11, In IEEE Xplore*, 2011.
14. M.K. Hani, H.Y. Wen, and A. Paniandi. Design and implementation of a private and public key crypto processor for next-generation it security applications. *Malaysia Journal of Comp. Science*, 19(1):29–45, 2006.

15. T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, and C. Irvine. Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems. *IEEE Symposium on Security and Privacy*, pages 281–295, 2007.
16. K. Kepa, F. Morgan, K. Kosciuszkiewicz, and T. Surmacz. Serecon: a secure reconfiguration controller for self-reconfigurable systems. *International Journal of Critical Computer-Based Systems*, pages 86–103, 2010.
17. M. Machhout, Z. Guitouni, K. Torki, L. Khriji, and R. Tourki. Coupled FPGA/ASIC Implementation of Elliptic Curve Crypto-Processor. *International Journal of Network Security & Its Applications*, 2(2):100–112, 2010.
18. J. M. McConnell. TEMPEST/2-95. *NSTISSAM*, 1995. http://cryptome.org/tempest-2-95.htm.
19. M.M. Parelkar and K. Gaj. Implementation of eax mode of operation for fpga bitstream encryption and authentication. In *FPT'05*, pages 335–336, 2005.
20. K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede. Multicore curve-based cryptoprocessor with reconfigurable modular arithmetic logic units over GF (2^n). *IEEE TC*, pages 1269–1282, 2007.
21. E. Simpson and P. Schaumont. Offline hw/sw authentication for reconfigurable platforms. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 311–323, 2006.
22. F.X. Standaert, L. van Oldeneel tot Oldenzeel, D. Samyde, and J.J. Quisquater. Power analysis of FPGAs: How practical is the attack? *FPL'03*, pages 701–711, 2003.
23. Xilinx. *UG702: Partial Reconfiguration User Guide*, 2010. http://www.xilinx.com/.
24. Xilinx. *XAPP1105: Single Chip Crypto Lab Using PR/ISO Flow with the Virtex-5 Family for ISE Design Suite 12.1*, 2011. http://www.xilinx.com/.